Akansha Balhara

- Roll no.: 1/23/SET/BCS/433
- Date : 12/02/2026
- Exp no. : 5

```python
import numpy as np

# ============================================================
# 1. XOR Dataset
# ============================================================
X_in = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
], dtype=float)

Y_true = np.array([[0], [1], [1], [0]], dtype=float)

# ============================================================
# 2. Activation Functions (ReLU + Sigmoid)
# ============================================================
def relu(z):
    return np.maximum(0, z)

def relu_grad(z):
    return (z > 0).astype(float)

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sigmoid_grad(a):
    return a * (1 - a)

# ============================================================
# 3. Network Structure
# ============================================================
np.random.seed(2024)

inp = 2
h1 = 4
h2 = 3
out = 1

# He initialization for ReLU layers
W1 = np.random.randn(inp, h1) * np.sqrt(2 / inp)
b1 = np.zeros((1, h1))

W2 = np.random.randn(h1, h2) * np.sqrt(2 / h1)
```

```
b2 = np.zeros((1, h2))

# Xavier initialization for sigmoid output
W3 = np.random.randn(h2, out) * np.sqrt(1 / h2)
b3 = np.zeros((1, out))

lr = 0.08
epochs = 15000
clip_val = 1.0  # for gradient clipping

# =============================================================
# 4. Training Loop
# =============================================================
for epoch in range(epochs):

    # ----------------- Forward Pass -----------------
    Z1 = X_in @ W1 + b1
    A1 = relu(Z1)

    Z2 = A1 @ W2 + b2
    A2 = relu(Z2)

    Z3 = A2 @ W3 + b3
    A3 = sigmoid(Z3)   # final prediction

    # ----------------- Binary Cross Entropy Loss -----------------
    loss = -np.mean(Y_true * np.log(A3 + 1e-9) + (1 - Y_true) *
np.log(1 - A3 + 1e-9))

    # ----------------- Backpropagation -----------------
    dZ3 = A3 - Y_true
    dW3 = A2.T @ dZ3
    db3 = np.sum(dZ3, axis=0, keepdims=True)

    dA2 = dZ3 @ W3.T
    dZ2 = dA2 * relu_grad(Z2)
    dW2 = A1.T @ dZ2
    db2 = np.sum(dZ2, axis=0, keepdims=True)

    dA1 = dZ2 @ W2.T
    dZ1 = dA1 * relu_grad(Z1)
    dW1 = X_in.T @ dZ1
    db1 = np.sum(dZ1, axis=0, keepdims=True)

    # ----------------- Gradient Clipping -----------------
    for grad in [dW1, dW2, dW3]:
        np.clip(grad, -clip_val, clip_val, out=grad)

    # ----------------- Parameter Updates -----------------
    W3 -= lr * dW3
    b3 -= lr * db3

    W2 -= lr * dW2
    b2 -= lr * db2

    W1 -= lr * dW1
    b1 -= lr * db1

    # ----------------- Logs -----------------
    if epoch % 2000 == 0:
        print(f"Epoch {epoch} | BCE Loss: {loss:.5f}")

# ============================================================
# 5. Final Testing
# ============================================================
print("\n=========== FINAL XOR RESULTS ===========\n")

for sample in X_in:
    s1 = relu(sample @ W1 + b1)
    s2 = relu(s1 @ W2 + b2)
    outp = sigmoid(s2 @ W3 + b3)

    print(f"Input {sample} → Output {outp[0][0]:.4f} → Class
{int(outp[0][0] > 0.5)}")
```

```
Epoch 0 | BCE Loss: 0.74113
Epoch 2000 | BCE Loss: 0.00191
Epoch 4000 | BCE Loss: 0.00091
Epoch 6000 | BCE Loss: 0.00060
Epoch 8000 | BCE Loss: 0.00044
Epoch 10000 | BCE Loss: 0.00035
Epoch 12000 | BCE Loss: 0.00029
Epoch 14000 | BCE Loss: 0.00025

=========== FINAL XOR RESULTS ===========

Input [0. 0.] → Output 0.0009 → Class 0
Input [0. 1.] → Output 1.0000 → Class 1
Input [1. 0.] → Output 1.0000 → Class 1
Input [1. 1.] → Output 0.0000 → Class 0
```

```
import numpy as np

# =======================================================
# Utility: Activation Functions
# =======================================================
def relu(z):
    return np.maximum(0, z)

def relu_grad(z):
    return (z > 0).astype(float)
```

```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def sigmoid_grad(a):
    return a * (1 - a)


# ========================================================
# XOR Neural Network (OOP Design with Momentum)
# ========================================================
class XORNet:

    def __init__(self, lr=0.1, momentum=0.9):
        np.random.seed(999)

        # Architecture: 2 → 16 → 8 → 1
        self.W1 = np.random.randn(2, 16) * np.sqrt(2 / 2)
        self.b1 = np.zeros((1, 16))

        self.W2 = np.random.randn(16, 8) * np.sqrt(2 / 16)
        self.b2 = np.zeros((1, 8))

        self.W3 = np.random.randn(8, 1) * np.sqrt(1 / 8)
        self.b3 = np.zeros((1, 1))

        # Momentum terms
        self.V1 = np.zeros_like(self.W1)
        self.V2 = np.zeros_like(self.W2)
        self.V3 = np.zeros_like(self.W3)

        self.lr = lr
        self.momentum = momentum

    # --------------------------------------------------------
    # Forward pass
    # --------------------------------------------------------
    def forward(self, X):
        self.Z1 = X @ self.W1 + self.b1
        self.A1 = relu(self.Z1)

        self.Z2 = self.A1 @ self.W2 + self.b2
        self.A2 = relu(self.Z2)

        self.Z3 = self.A2 @ self.W3 + self.b3
        self.A3 = sigmoid(self.Z3)

        return self.A3

    # --------------------------------------------------------
    # Backward pass + Momentum update
    # ------------------------------------------------------
    def backward(self, X, Y_true):
        m = len(X)

        dZ3 = (self.A3 - Y_true)
        dW3 = self.A2.T @ dZ3 / m
        dB3 = np.sum(dZ3, axis=0, keepdims=True) / m

        dA2 = dZ3 @ self.W3.T
        dZ2 = dA2 * relu_grad(self.Z2)
        dW2 = self.A1.T @ dZ2 / m
        dB2 = np.sum(dZ2, axis=0, keepdims=True) / m

        dA1 = dZ2 @ self.W2.T
        dZ1 = dA1 * relu_grad(self.Z1)
        dW1 = X.T @ dZ1 / m
        dB1 = np.sum(dZ1, axis=0, keepdims=True) / m

        # ---------------- MOMENTUM UPDATE ----------------
        self.V3 = self.momentum * self.V3 - self.lr * dW3
        self.W3 += self.V3
        self.b3 -= self.lr * dB3

        self.V2 = self.momentum * self.V2 - self.lr * dW2
        self.W2 += self.V2
        self.b2 -= self.lr * dB2

        self.V1 = self.momentum * self.V1 - self.lr * dW1
        self.W1 += self.V1
        self.b1 -= self.lr * dB1

    # ------------------------------------------------------
    # Loss (Binary Cross Entropy)
    # ------------------------------------------------------
    def compute_loss(self, y):
        eps = 1e-9
        return -np.mean(y * np.log(self.A3 + eps) + (1 - y) * np.log(1
- self.A3 + eps))

    # ------------------------------------------------------
    # Training with Early Stopping + LR Decay
    # ------------------------------------------------------
    def train(self, X, Y, epochs=12000, decay=0.999):
        best_loss = float("inf")
        patience = 600
        counter = 0

        for i in range(epochs):
            Y_pred = self.forward(X)
            loss = self.compute_loss(Y)
            self.backward(X, Y)

            # learning rate decay
            self.lr *= decay

            # logging
            if i % 1500 == 0:
                print(f"[Epoch {i:5d}]  Loss = {loss:.6f}  LR =
{self.lr:.5f}")

            # EARLY STOPPING
            if loss < best_loss:
                best_loss = loss
                counter = 0
            else:
                counter += 1

            if counter >= patience:
                print(f"\nEarly Stop Triggered at Epoch {i}")
                break


# ========================================================
# 6. RUN NETWORK
# ========================================================
X = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=float)
Y = np.array([[0],[1],[1],[0]], dtype=float)

model = XORNet(lr=0.15, momentum=0.85)
model.train(X, Y)

# ========================================================
# 7. FINAL TEST
# ========================================================
print("\n=========== FINAL XOR PREDICTIONS ===========\n")
for sample in X:
    pred = model.forward(sample.reshape(1, -1))
    print(f"Input {sample} → Output {pred[0][0]:.4f} → Class:
{int(pred[0][0] > 0.5)}")

[Epoch     0]  Loss = 0.719454  LR = 0.14985
[Epoch  1500]  Loss = 0.000554  LR = 0.03341
[Epoch  3000]  Loss = 0.000413  LR = 0.00745
[Epoch  4500]  Loss = 0.000390  LR = 0.00166
[Epoch  6000]  Loss = 0.000385  LR = 0.00037
[Epoch  7500]  Loss = 0.000384  LR = 0.00008
[Epoch  9000]  Loss = 0.000384  LR = 0.00002
[Epoch 10500]  Loss = 0.000384  LR = 0.00000
```

```
=========== FINAL XOR PREDICTIONS ===========

Input [0. 0.] → Output 0.0013 → Class: 0
Input [0. 1.] → Output 0.9999 → Class: 1
Input [1. 0.] → Output 0.9999 → Class: 1
Input [1. 1.] → Output 0.0001 → Class: 0
```